A Comparative Study of Neural Network Pruning Strategies for AI Deployment on Edge Devices

Amirhossein Douzandeh Zenoozi[1 [0009−0002−2839−1202]], Laura Erhan[1 [0000−0001−7727−784X]], Antonio Liotta[1 [0000−0002−2773−4421]], and Lucia Cavallaro[2 [0000−0002−2367−6084]]

[1] Free University of Bozen-Bolzano, Bolzano, Italy,
{adouzandehzenoozi,laura.erhan,antonio.liotta}@unibz.it,
[2] Radboud University, Nijmegen, Netherlands,
lucia.cavallaro@ru.nl,

Abstract. Artificial neural networks have become crucial across fields like IoT, computer vision, and medicine. Their use for a variety of industrial applications, and the ever-expanding IoT, contributed to a growing interest for lightweight neural network models suitable for deployment in environments with limited computational capabilities. Pruning techniques aim to address the computational and storage demands of these models. In this work, we investigate the impact that different pruning methods have on Multi-Layer Perceptron (MLP) networks. We compare pre-training, in-training, post-training, and the SET-Method pruning approaches while considering a variety of parameters. We find that highly sparse small-scale MLPs can achieve accuracies similar to their fully connected counterparts. Furthermore, energy consumption and inference time are primarily influenced by model size, rather than sparsity levels. This research provides insight into optimizing and further understanding neural networks and their applicability to real-world applications.

Keywords: Multi-Layer Perceptron, Sparse Neural Networks, Pruning Techniques, Sparse Evolutionary Training, Graph Theory

1 Introduction

With the advancement of deep learning, artificial neural networks (ANNs) have been commonly used in several domains including the Internet of Things (IoT) [3], computer vision [6,11], speech recognition [1], and medicine [12,13].

In recent years, we witnessed a rapid growth of the use of Artificial Intelligence (AI) in industry. Combining AI with the multitude of IoT devices found at the Edge, became of interest and is driven by the need and desire to deploy and run AI on lightweight devices to keep and process the data close to its source, saving energy costs, minimising the transfer of information, and bandwidth usage [9]. These edge devices have fewer computational resources compared to high-performance computers (HPC), and usually cannot just run heavy models

out of the box. Pruning neural network models [15] can be a viable solution to address this challenge of deploying at the edge. Gaining insight into how different pruning methods affect the behavior of a model can serve as the best guide for choosing the most effective approach. Comparing different pruning techniques while monitoring a set of key parameters becomes crucial as a first step of analysis.

In this paper, we investigate the behavior of two Multi-Layer Perceptron (MLP) networks [14], namely a small-scale one which has 826 neurons and a large-scale one with 9,794 neurons, when sparsified. We sparsify the two models by applying four different pruning methods: (i) pre-training pruning, in which the training begins with a randomly pruned network; (ii) in-training pruning, where during training and after each round only the stronger connections between neurons in each layer of the model are chosen and kept; (iii) post-training pruning, where the model is pruned after the training is completed; (iv) Sparse Evolutionary Training (SET) [10], where the network is pruned before the training phase based on the *Erdős-Rényi* [2] model as a first step and, during the training, a fraction of the smallest positive and largest negative weights are replaced with new random ones at each epoch. For the comparison of the four approaches we look at the collected metrics, namely training time, testing accuracy, F1 scores, average inference time, estimated total energy consumption, model size, and peak memory usage.

In Mocanu *et al.*'s [10] work on SET, the authors present a method to reduce the number of connections between neurons in fully connected networks by using adaptive sparse connectivity, starting from a neural network initialized through the *Erdős-Rényi* model. This approach is similar to the in-training approach. The main difference between the two is given by the initialization phase, namely using a *Random Unstructured* for the in-training pruning versus the *Erdős-Rényi* random network in the case of SET. In this paper, we offer a direct comparison and evaluation of the more intricate SET versus the simple in-training approach.

Furthermore, in Kalyanam *et al.*'s [7] work on unstructured pruning for Multi-Layer Perceptrons (MLPs) with *tanh* activation, the authors introduce a new way to remove unnecessary neurons by using the *tanh* function, which is centered around zero. Their method focuses on pruning neurons based on how active they are, especially those with activations near -1, 0, or 1. This is different from traditional methods that usually remove neurons based on their connection strengths or predefined rules. While their research on the MNIST dataset achieves about 32% sparsity with only a 5.6% drop in accuracy, our approach applies over 60% sparsity while maintaining accuracy levels close to that of a fully connected model. In addition, in Kalyanam *et al.*'s work, the results are limited to a single model and two datasets, which limits the wide comparison of their pruning approach across different models and datasets. We tackle model limitations by also comparing the behaviour of a large and small-scale MLP, and the same dataset (*i.e.*, MNIST).

Frankle *et al.* [4] introduced the Lottery Ticket Hypothesis, which shows that within large, dense neural networks, there are smaller sub-networks ("winning

tickets") that can be trained to perform as good as the fully-connected model. Their method involves pruning after training, which can be resource-intensive. Our research not only explores the post-training method, but we also compare those performances with two other different pruning methods (*i.e.*, before and during training), which provides more flexibility. Notably, while Frankle *et al.* applied this hypothesis to the *Lenet-300-100* model, which contains 1,194 neurons and 266,200 connections [8] in the fully-connected form, our research involves a comparative study of two different-sized models, one of them is larger than *Lenet-300-100* and the other one is smaller than that on the same dataset (*i.e.*, MNIST). This comparison leads to a better insight into the pruning approaches of the MLP models.

While our study primarily focuses on MLP pruning methods, other approaches such as Gradient Signal Preservation (Grasp) [16], and one-shot magnitude pruning (OMP) [18] offer valuable insights into model sparsity. Although not directly compared in this work, they represent important alternatives that may be investigated further in future research, especially when scaling to larger, more complex models and different architectures, since the main part of their research is investigating the pruning approaches of the convolutional neural networks (CNN) models.

This work aims to provide a comparison and evaluation of four different pruning approaches for two MLP models, a small-scale and a large-scale one, in order to identify how much we can sparsify a model, while still maintaining accuracies comparable to their fully-connected counterpart. Furthermore, we investigate and discuss the possible benefits of employing such techniques for a possible IoT Edge deployment. While all methods can achieve accuracies comparable to the not sparse network, SET further pushes the sparsity level at which an accuracy drop occurs. Small-scale MLP models can keep up accuracy-wise with their large counterparts while employing a much-reduced number of neurons (*i.e.*, 826 neurons for the small-scale and 9,794 neurons for the large-scale) and connections (*i.e.*, 3,888, and 3,352,800 connections for the small-scale and large-scale respectively with 70% sparsity).

2 Methodology

2.1 Dataset

We aim to analyze and better understand the behavior of the MLP model when different pruning strategies are used to train our model. For this, it is important to use datasets that are not only publicly available but also relatively simple compared to others. The MNIST dataset, which is a well-known benchmark in machine learning and computer vision, meets all these criteria.

MNIST is a large collection of handwritten digit grayscale images (*i.e.*, 10 classes to be identified) used for training and testing image processing and machine learning systems. This dataset provides a baseline for comparison when leading research on different model architectures and it is one of the existing built-in datasets over all artificial neural network libraries.

2.2 Considered pruning methods

To better understand pruning techniques, it is important to first examine fully-connected networks. In a classic training process of the MLP networks, each neuron in one layer is connected to all neurons in the subsequent layer, which is why these networks are named as fully-connected. Each of these connections has an associated weight, and after training, the model uses these weights to make decisions by identifying the best match between the input and the output. Figure 1 [1] illustrates the pruning process. In the following paragraphs, a detailed description of the pruning techniques that were compared in this study is provided.

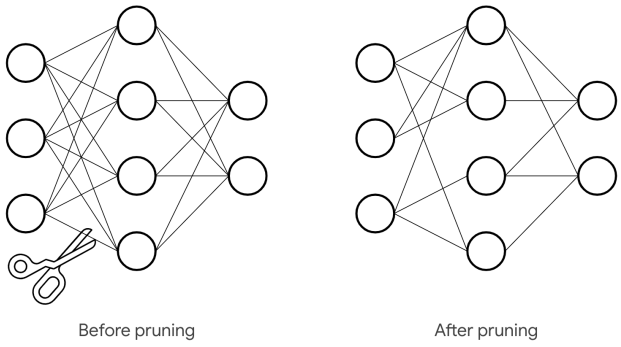

Fig. 1. The difference between fully-connected and pruned networks (Original figure from TensorFlow).

Pre-Training Approach is one of the simplest pruning methods, which employs the *L1 Unstructured* technique to remove weak connections immediately after creating the model and before starting the training phase. The advantage of using this method is its simplicity in development. Also, by starting with a pruned network, the model has fewer parameters to optimize, which can reduce computational costs and speed up the training phase.

In-Training Method is the second pruning method that employs the built-in *Random Unstructured* technique available in the PyTorch library to generate the initial model. During training, at each epoch, the weakest connections are replaced with other random connections that were previously pruned from the network. This method not only has the advantages of the *Pre-Training Method* but also, generates new connections to find the best connections with stronger weights in each layer and each training round.

[1] Image by TensorFlow. Available at https://blog.tensorflow.org/2019/05/tf-model-optimization-toolkit-pruning-API.html

Post-Training Approach is the simplest technique among the ones herein described. This method prunes the network after the training, making it possible to obtain different configurations without the need to train the model multiple times. By starting with a fully-connected version of the model, we can generate pruned networks by simply removing the smallest connections after the training phase is completed. However, while we do not need to retrain the network to achieve different sparsity levels, this approach does have some drawbacks. For instance, the model does not have time to optimize the weights during the training. In addition, if the model overfits before pruning, the pruned versions may still suffer from overfitting.

SET approach [10] differs from the *Random Unstructured* or *L1 Unstructured* techniques for generating the initial model by using the *Erdős-Rényi* model [2]—one of the most famous random models in graph theory. In our case, it allows us to generate an initial sparse network. To do so, we have to calculate the probability for each connection between each pair of neurons, as later explained in more detail. Additionally, this method adopts the same approach as the *In-Training Method* to optimize parameters during training. Specifically, it replaces the smallest positive and largest negative connections with those that have been removed from the network. This ensures that in each round, the model optimizes the largest existing weights, leading to a network with the strongest connections by the end of the training phase.

Equation 1 illustrates how to calculate the probability of creating a link between pairs of neurons. In this formula, ϵ represents the sparsity level, while n^k, and n^{k-1} represent the number of input and output neurons of the layer, respectively.

$$p = \frac{\epsilon \times (n^k + n^{k-1})}{n^k \times n^{k-1}} \tag{1}$$

In addition, Equation 2 presents the percentage form of ϵ which is calculated based on the number of input and output neurons of the network. This equation helps in having a more readable form of the sparsity level, making it suitable for any type of network regardless of the architecture and number of neurons.

$$\epsilon = \frac{\text{percentage} \times (n^k \times n^{k-1})}{n^k + n^{k-1}} \tag{2}$$

2.3 Experimental Setup

MLP model The main reason for choosing the MLP architecture is that in this type of network, the connections between neurons are not only straightforward—each neuron in a layer has a directed path to neurons in the next layer—but also each connection has a single float weight. In contrast, complex networks like ResNet-based architectures have additional residual connections [5], and the convolutional layers use matrices as weights in the feature extraction part of the network.

We begin our analysis with a small-scale MLP model, which is a 4-layer network comprising a total of 826 neurons. This includes 784 input neurons and two hidden layers with 16 neurons each. The fully-connected version of this model has 12,960 connections, which is the minimum number of connections required for a fully-connected network to achieve an accuracy of up to 90% experimentally.

For a better comparison with a larger architecture, we also used a larger MLP model. This model is composed of 4,000 neurons in the first hidden layer, 1,000 neurons in the second hidden layer, and again 4,000 in the last hidden layer. The fully-connected version of this model has 11,176,000 connections. The structure of the large MLP is the same as in the work of Mocanu *et al.* [10], namely the number of neurons in hidden layers, and the order of the hidden layers. For both models, we used a learning rate of *1e-3*, a batch size of 4, and trained the networks 3 times for 60 epochs.

Dataset split For each phase of training, validation, and testing, we use 80%, 10%, and 10% of the total dataset, respectively, which equals to 12,000 samples for the 80% and 1,500 samples for the 10% fractions.

Sparsification In this experiment, we investigate the model's performance with four different pruning techniques: (i) pre-training, (ii) in-training, (iii) post-training, (iv) and the SET approach. These are described in detail in the previous section 2.2.

Evaluation metrics To evaluate the model performances when the four pruning techniques are used, we considered nine metrics, namely: F1 score, recall, precision, test accuracy, loss, training energy, training time, average inference time, and maximum memory usage. F1 Score, recall, precision, and accuracy help measure the model's performance after the training phase. Training energy, memory usage, and training time lead to a better insight into the needed computational resources. Monitoring loss is crucial to prevent overfitting issues.

Deployment and implementation We trained our model three times and calculated the average values for all these metrics to make sure that the results were reliable and the experiment's environment had not been affected by any unwanted parameters. To implement the neural network model, several libraries can be used, including TensorFlow and PyTorch, or even developing from scratch without any libraries. We opted for the PyTorch library to develop the MLP network due to its flexibility. It allows us to utilize not only the built-in PyTorch functions but also to create custom functions as needed.

For the model training, we utilized a high-performance computing (HPC) system, which provided an isolated experimental environment for each training session. Specifically, we used the **NVIDIA A100-SXM4-80GB** GPU. To monitor energy consumption during the training process, we used the **Zeus**

Project [2]. Although the Zeus tool is straightforward to use, it presents some limitations in HPC environments. Notably, when using HPC systems, the GPU resources are often partitioned, and Zeus cannot measure energy consumption for individual GPU partitions. In our comparative experiment, training the model with the same configuration on a **Lenovo Legion 5**, equipped with an **NVIDIA GeForce RTX 3060** GPU and **16GB RAM**, required approximately half the energy, on average, compared to the HPC system.

3 Results

In this section, we discuss the results of our experiments for each pruning method and each model (the small-scale and the larger one), as well as for the five different sparsity levels, which are 60%, 70%, 80%, 85%, and 90%. These are depicted in Tables 1- 4 and are the average results after three times training the model with the same configuration for a number of 60 epochs.

Table 1. Pre-Training Approach

Sparsity Level	Training Time (s)	Test Accuracy (%)	F1 Score Test	Avg. Inference Time (s)	Total Energy (J)
Small-Scale MLP					
0%	1156.7	94.28	0.9417	1.43e-4	84559.65
60%	1302.4	92.45	0.9232	1.47e-4	87875.22
70%	1413.6	91.72	0.9155	1.50e-4	95167.86
80%	1306.9	89.50	0.8936	1.45e-4	87815.67
85%	1305.0	87.13	0.8689	1.52e-4	95696.80
90%	1307.1	56.03	0.4852	1.47e-4	87834.71
Large-Scale MLP					
0%	1445.4	96.88	0.9683	2.61e-4	128324.69
60%	1914.9	94.62	0.9452	2.80e-4	192746.78
70%	1609.3	93.70	0.93557	3.32e-4	209751.68
80%	1704.8	92.28	0.9212	2.60e-4	210908.06
85%	1812.0	90.63	0.9040	2.68e-4	213176.28
90%	1632.3	64.72	0.5737	2.64e-4	151085.52

Table 1 presents the evaluation metrics of the two models when the pruning method is applied before the training phase, immediately after initializing the model. It is interesting to note that for both models, the performance at 90% sparsity is low, with accuracies below 65%.

Table 2 shows the models' behavior when pruning is applied not only after initialization but also continuously during training, where the smallest positive and largest negative weights are replaced with new random connections to discover better potential connections from the pruned ones. For the small-scale network,

[2] Available at https://ml.energy/zeus/

Table 2. In-Training Approach

Sparsity Level	Training Time (s)	Test Accuracy (%)	F1 Score Test	Avg. Inference Time (s)	Total Energy (J)
Small-Scale MLP					
0%	1156.7	94.28	0.9417	1.43e-4	84559.65
60%	1301.3	93.68	0.9357	1.44e-4	87527.39
70%	1450.4	92.93	0.9281	1.44e-4	97640.40
80%	1303.3	91.77	0.9162	1.47e-4	87561.64
85%	1352.7	86.75	0.8599	1.50e-4	95475.16
90%	1303.1	85.53	0.8946	1.41e-4	97324.14
Large-Scale MLP					
0%	1445.4	96.88	0.9683	2.61e-4	128324.69
60%	1920.4	94.88	0.9479	2.69e-4	229930.56
70%	1798.7	93.75	0.9362	3.61e-4	219469.84
80%	1725.8	91.73	0.9154	2.62e-4	201693.98
85%	1971.4	89.67	0.8941	2.89e-4	192052.66
90%	1655.3	51.18	0.4380	2.70e-4	152885.82

the accuracy gradually decreases as sparsity increases. At 60% sparsity, the test accuracy remains relatively stable at 93.68%, with a slight drop compared to the fully-connected model (*i.e.*, 94.28%). However, as sparsity reaches 85%, test accuracy drops to 86.75%, indicating a loss in performance. In contrast, for the large-scale network, the model maintains test accuracies above 89% for up to 85% sparsity. However, at 90% sparsity, the performance rapidly declines, with accuracy dropping to 51.18%. However, the small-scale model is more robust and accuracy remains high also for 90% sparsity level, with 85.53%.

Results from the post-training pruning method are shown in Table 3 in which there is a noteworthy difference between small-scale and large-scale models. For example, with small-scale models, we were unable to achieve an accuracy greater than 80% after reaching 60% sparsity. In contrast, with large-scale models, we were able to increase the sparsity from 60% to 85% while still maintaining an accuracy above 80%.

Lastly, the outcomes achieved in training the network accordingly with the SET approach are displayed in Table 4. Here, the accuracy trend is similar to the one observed with the In-Training Approach. Additionally, as the model size increases, we observe a more consistent and reliable pattern in both accuracy and F1 score compared to the small-scale model. However, we notice once again that the small-scale model is more robust also at higher sparsity levels (90%). Furthermore, the performance of this method is close to that of the In-Training Approach. For example, with 85% sparsity applied to the large-scale model, the accuracy reaches 89% using the In-Training approach, whereas the SET approach achieves 90% at the same sparsity level. In contrast, we expected that the impact of using the *Erdős-Rényi* model for the initialization phase would result in even greater improvements.

Table 3. Post-Training Approach

Sparsity Level	**Training Time (s)**	**Test Accuracy (%)**	**F1 Score Test**	**Avg. Inference Time (s)**	**Total Energy (J)**
Small-Scale MLP					
0%	1156.7	94.28	0.9417	1.43e-4	84559.65
60%	1175.0	41.83	0.4136	1.44e-4	78933.68
70%	1369.8	24.27	0.1630	1.51e-4	91942.87
80%	1198.2	17.72	0.0797	1.47e-4	80718.98
85%	1351.3	15.92	0.0582	1.50e-4	90808.66
90%	1177.3	13.53	0.0466	1.42e-4	118645.57
Large-Scale MLP					
0%	1445.4	96.88	0.9683	2.61e-4	128324.69
60%	1735.3	96.23	0.9617	2.74e-4	201965.11
70%	1614.6	94.87	0.9477	3.57e-4	182312.32
80%	1551.8	89.50	0.8944	2.62e-4	179848.32
85%	1760.8	81.43	0.8053	3.11e-4	141373.65
90%	1488.6	68.82	0.6522	2.64e-4	130884.10

Generally, as expected, increasing the size of the models leads to an increase in average inference time across all pruning approaches. A similar pattern can be noticed in energy consumption during the training phase, as larger models do require more energy. However, changing the sparsity-inducing approach during the training phase in both small-scale and large-scale networks does not result in significant changes. The small-scale MLP is more robust when dealing with higher sparsity levels for the SET and in-training approaches. Furthermore, pruning at the beginning of the training is better in terms of performance than pruning at the end of the training for a small neural network which may be appropriate for an IoT Edge scenario. Depending on the use case scenario and the desired performance levels, even simple pruning techniques can prove to be valuable, such as pre-training pruning. If accuracy is the most important parameter we consider, then SET should be the choice.

To have a better insight into the amount of energy consumption for the training, we use a simple comparison. A 100-watt light bulb consumes 0.1 kWh per hour. According to the energy equation, $Energy = Power \times Time$, on average, the total energy used for training the large-scale network would be enough to power this bulb for approximately 21 minutes. Also, by measuring the energy for loading the model, we see that higher sparsity levels slightly reduce the energy needed to load the model. Moreover, in all tables, the F1 score aligns with the behavior of accuracy.

It is important to note that the hardware used in this study, including NVIDIA GPUs and the PyTorch framework, may not fully leverage the benefits of model sparsity during computation. Sparse matrices are not inherently optimized on this hardware, which could result in similar training and inference times across different sparsity levels [17]. This potential limitation may impact

the full realization of the speed and efficiency gains that typically come from pruning in more specialized environments.

Table 4. SET Approach

Sparsity Level	**Training Time (s)**	**Test Accuracy (%)**	**F1 Score Test**	**Avg. Inference Time (s)**	**Total Energy (J)**
Small-Scale MLP					
0%	1156.7	94.28	0.9417	1.43e-4	84559.65
60%	1309.2	93.90	0.9380	1.43e-4	87733.54
70%	1475.5	93.48	0.9342	1.70e-4	99115.68
80%	1305.0	87.28	0.8708	1.47e-4	87714.46
85%	1470.7	88.87	0.8863	1.54e-4	98769.88
90%	1288.0	87.08	0.8678	1.46e-4	131385.12
Large-Scale MLP					
0%	1445.4	96.88	0.9683	2.61e-4	128324.69
60%	1944.2	94.97	0.9487	3.02e-4	227991.83
70%	1756.0	93.50	0.9335	2.78e-4	181478.66
80%	1711.2	92.03	0.9185	2.62e-4	210246.38
85%	1936.6	90.08	0.8985	2.82e-4	191658.62
90%	1736.2	70.82	0.6249	2.63e-4	182436.73

4 Conclusion and future works

In this work, we compare and evaluate different pruning methods for two MLP models of different sizes. Selecting the best pruning technique depends on the model's architecture and metrics of interest. For example, as the model size increases, performance tends to improve even with the same level of sparsity, though the small performance gain may not outweigh savings in model size. Additionally, both energy consumption and inference time are linked to model size, while sparsity percentage does not significantly affect these factors. Achieving high accuracy with sparse topologies in small-scale networks is promising for IoT edge device deployment.

While this study provides valuable insights, several areas remain unexplored, offering opportunities for further research, such as comparing more complex models in similar settings. Optimizing model savings and leveraging sparsity for reduced training and inference times is also of interest. Furthermore, using sparsification techniques in Federated Learning systems is another area for future research.

Acknowledgments

This work was supported by the PRIN 2020 projects COMMON-WEARS (Grant no. I53C21000210001) and SELF4COOP (Grant no. E53D23000910001).

References

1. Avula, H., Ranjith, S Pillai, A.: CNN based recognition of emotion and speech from gestures and facial expressions. In: 2022 6th International Conference on Electronics, Communication and Aerospace Technology. IEEE (2022)
2. Erdös, P., Rényi, A.: On random graphs i. Publicationes Mathematicae Debrecen **6**, 290–297 (1959)
3. Fanti Coelho Limaa, R., Segata, M., Azfar Yaqub, M., Liotta, A.: Minilearn: On-device learning for low-power iot devices. In: Proceedings of the 2023 International Conference on Embedded Wireless Systems and Networks, EWSN '23, p. 333–338. Association for Computing Machinery, New York, NY, USA (2023)
4. Frankle, J., Carbin, M.: The lottery ticket hypothesis: Finding sparse, trainable neural networks (2019). URL https://arxiv.org/abs/1803.03635
5. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition (2015). URL https://arxiv.org/abs/1512.03385
6. Islam, M.R., Zamil, M.Z.H., Rayed, M.E., Kabir, M.M., Mridha, M.F., Nishimura, S., Shin, J.: Deep learning and computer vision techniques for enhanced quality control in manufacturing processes. IEEE Access pp. 1–1 (2024)
7. Kalyanam, L.K., Katkoori, S.: Unstructured pruning for multi-layer perceptrons with tanh activation. In: 2023 IEEE International Symposium on Smart Electronic Systems (iSES), pp. 69–74 (2023). DOI 10.1109/iSES58672.2023.00025
8. Lecun, Y., Bottou, L., Bengio, Y., Haffner, P.: Gradient-based learning applied to document recognition. Proceedings of the IEEE **86**(11), 2278–2324 (1998). DOI 10.1109/5.726791
9. Li, E., Zeng, L., Zhou, Z., Chen, X.: Edge AI: On-demand accelerating deep neural network inference via edge computing. IEEE Trans. Wirel. Commun. **19**(1), 447–457 (2020)
10. Mocanu, D.C., Mocanu, E., Stone, P., Nguyen, P.H., Gibescu, M., Liotta, A.: Scalable training of artificial neural networks with adaptive sparse connectivity inspired by network science. Nat. Commun. **9**(1) (2018)
11. Moradi, M., Moradi, M., Palazzo, S., Rundo, F., Spampinato, C.: Image CAPTCHAs: When deep learning breaks the mold. IEEE Access **12**, 112,211–112,231 (2024)
12. Naji, M.F., Alzamil, F., Alajaj, M., Alshaqran, M., Alateeq, F.: Automatic COVID-19 diagnostic and classification intelligent system (ACDCIS). In: 2022 IEEE International Conference on Artificial Intelligence and Computer Applications (ICAICA). IEEE (2022)
13. Ondáš, S., Staš, J., Ševc, R.: Speech recognition as a supportive tool in the speech therapy game. In: 2024 34th International Conference Radioelektronika (RADIOELEKTRONIKA). IEEE (2024)
14. Popescu, M.C., Balas, V.E., Perescu-Popescu, L., Mastorakis, N.: Multilayer perceptron and neural networks. WSEAS Transactions on Circuits and Systems **8**(7), 579–588 (2009)
15. Voyant, C., Paoli, C., Nivet, M.L., Notton, G., Fouilloy, A., Motte, F.: Multi-layer perceptron and pruning. Turkish Journal of Physics (2017)
16. Wang, C., Zhang, G., Grosse, R.: Picking winning tickets before training by preserving gradient flow (2020). URL https://arxiv.org/abs/2002.07376
17. Yan, B., Root, A.J., Gale, T., Broman, D., Kjolstad, F.: Scorch: A library for sparse deep learning (2024). URL https://arxiv.org/abs/2405.16883

18. Zhang, Y., Yao, Y., Ram, P., Zhao, P., Chen, T., Hong, M., Wang, Y., Liu, S.: Advancing model pruning via bi-level optimization (2023). URL https://arxiv.org/abs/2210.04092